Louis XIV

AKA: State
So far, our object languages have been purely *functional*

- A function produces the same result every time for the same arguments
- That’s nice in some ways
- But that’s kind of limiting
- Sometimes we just need to keep track of changes
Non-Functional Procedure

(define counter 0)
(define (f x)
    (set! counter (+ x counter))
    counter)

• Using mutable variables to keep track of state
Non-Functional Procedure, now with boxes!

\[
\begin{align*}
(\text{define } f &= (\lambda (x)
\quad (\text{set-box}! \ b \ (+ \ x \ (\text{unbox} \ b)))
\quad (\text{unbox} \ b))))\
\end{align*}
\]

- Alternatively, can use mutable data structures
- Box \(\approx\) single-element mutable array
BFAE = FAE + Boxes

\[<\text{BFAE}> ::= <\text{num}> \]
| \{ + <\text{BFAE}> <\text{BFAE}> \}\]
| \{ - <\text{BFAE}> <\text{BFAE}> \}\]
| <\text{id}> \]
| \{ fun \{<\text{id}>\} <\text{BFAE}>\}\]
| \{ <\text{BFAE}> <\text{BFAE}> \}\]
| \{ newbox <\text{BFAE}> \}\]
| \{ setbox <\text{BFAE}> <\text{BFAE}> \}\]
| \{ openbox <\text{BFAE}> \}\]
| \{ seqn <\text{BFAE}> <\text{BFAE}> \}\]

\{with \{b \{newbox 0\}\}\]
\{ seqn \]
\{ setbox b 10 \}
\{ openbox b \}\}\]
\[\Rightarrow 10\]
Implementing Boxes with Boxes

(define-type BFAE-Value
  [numV (n number?)]
  [closureV (param-name symbol?)
    (body BFAE?)
    (ds DefSub?)]
  [boxV (container (box/c BFAE-Value?))])
Implementing Boxes with Boxes

; interp : BFAE? DefSub? -> BFAE-Value?
(define (interp a-bfae ds)
  (type-case BFAE a-bfae
    ...
    [newbox (val-expr)
      (boxV (box (interp val-expr ds)))]
    [setbox (box-expr val-expr)
      (set-box! (boxV-container
        (interp box-expr ds))
        (interp val-expr ds))]
    [openbox (box-expr)
      (unbox (boxV-container
        (interp box-expr ds)))])]

Nice parlor trick.
But we haven’t learned anything about how boxes work!